Andrés Felipe Pérez – 202215659
Rubén Darío Franco – 202012119
Mario Velásquez – 202020502

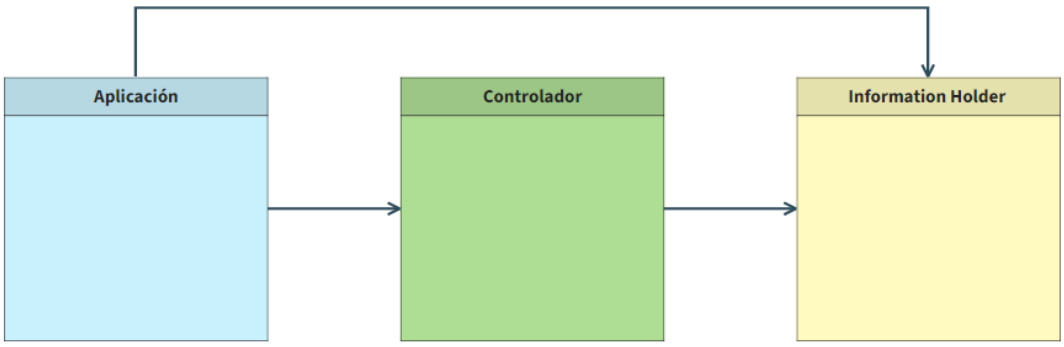
PROYECTO 1 – DPOO
ENTREGA 3
DOCUMENTO DE DISEÑO

DIAGRAMA DE CLASES:

Se desarrolló el siguiente diagrama de clases, se puede acceder a él a través del siguiente enlace:

https://app.moqups.com/ChEzWXm6MHnCurrl2Oqh4tlMnBExc1oC/view/page/a8bfb0d3e

Diagrama de alto nivel:



Nota: los colores de este diagrama se mantienen en los siguientes.

Diagrama de relaciones:

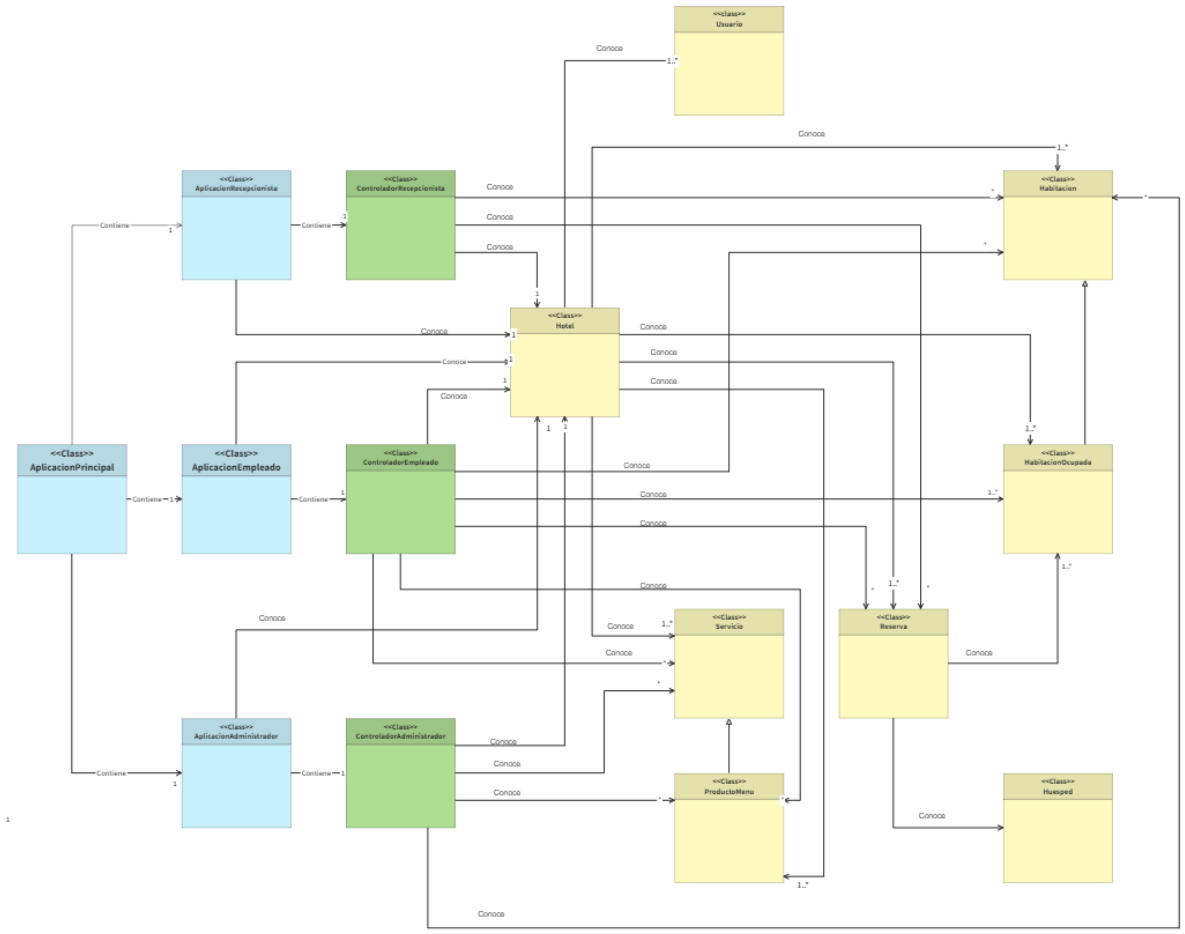
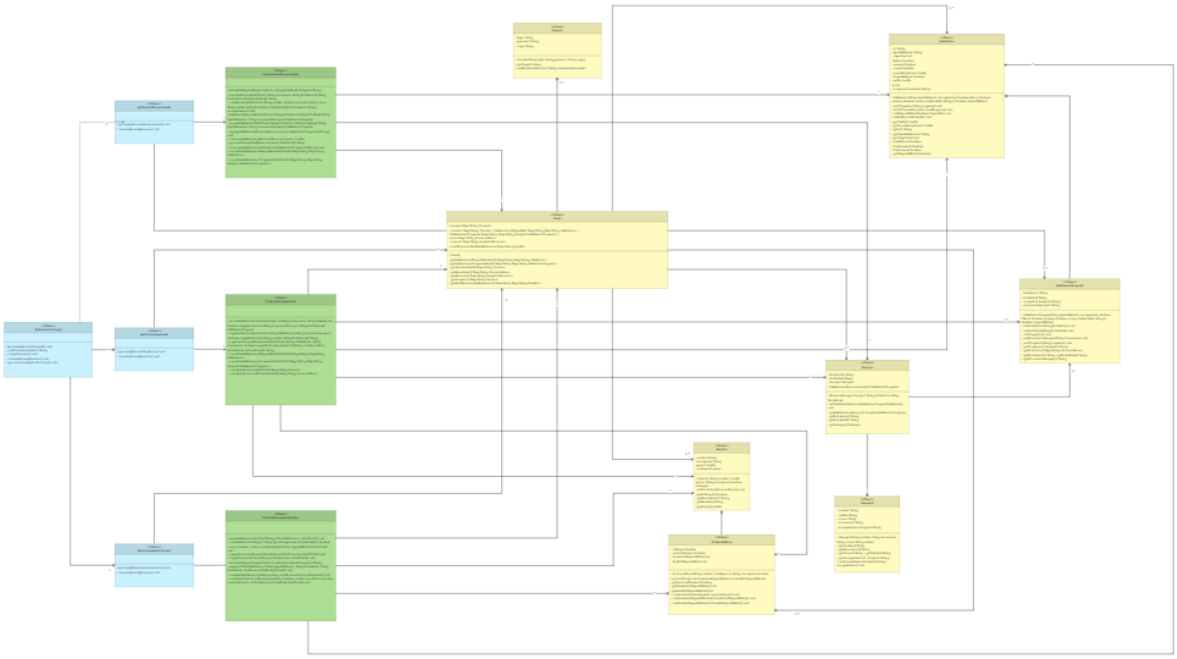
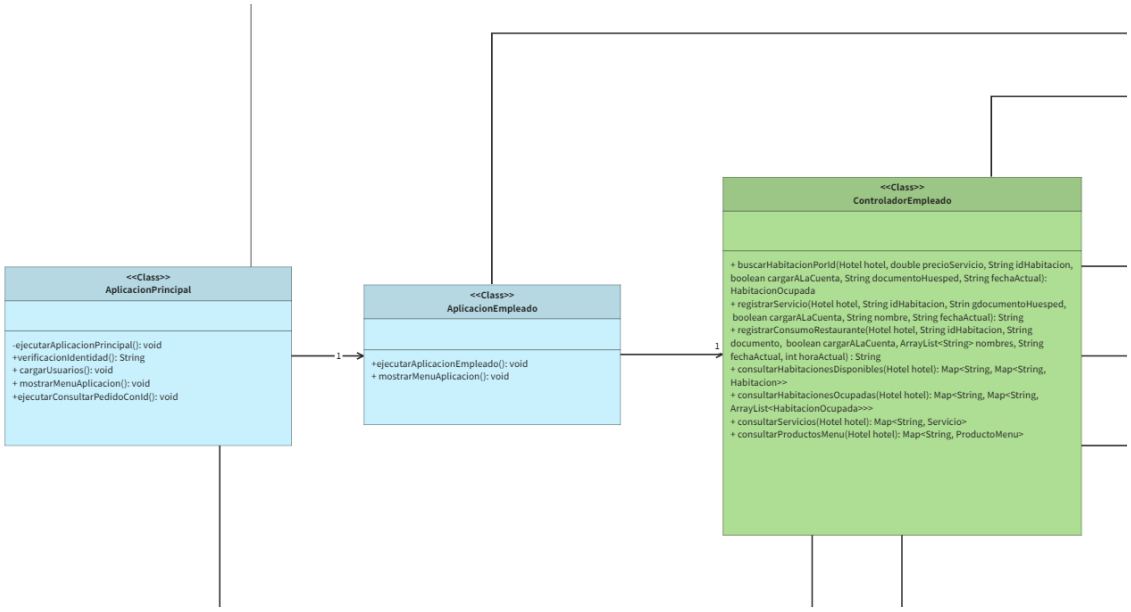
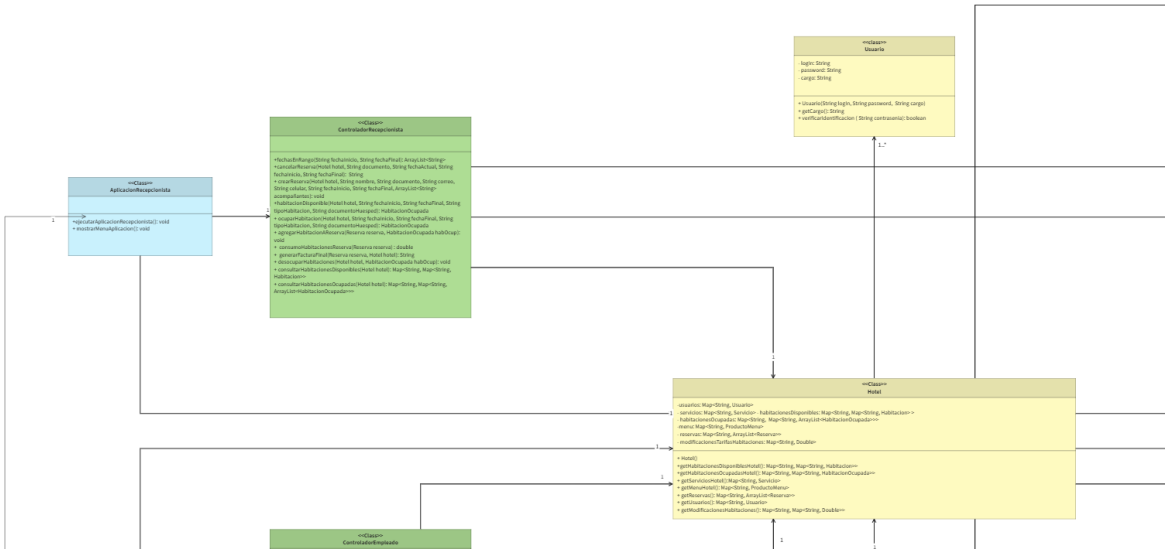
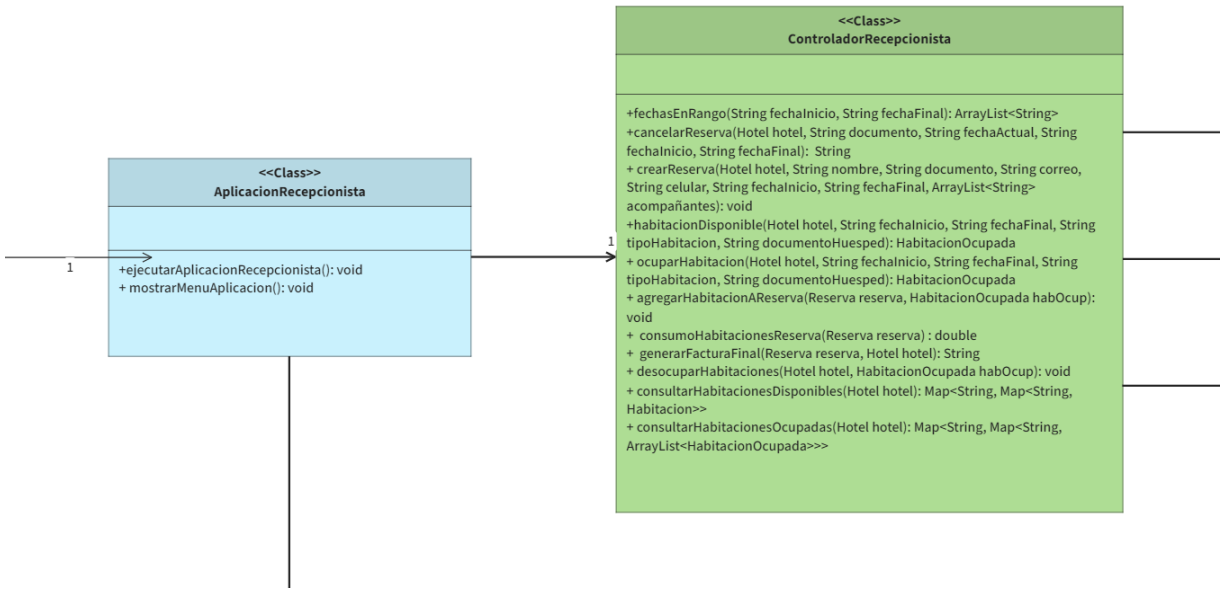


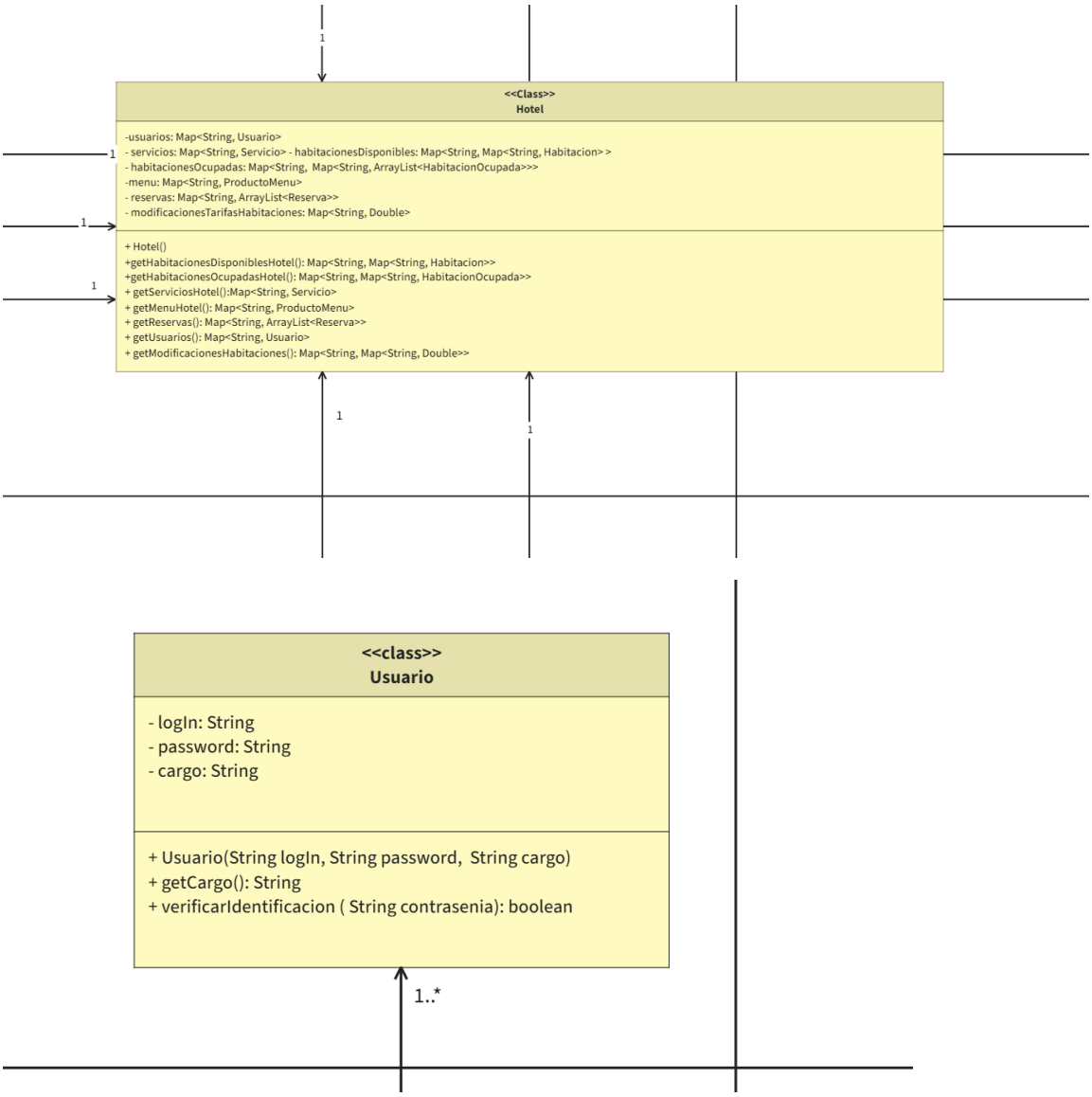
Diagrama de clases:

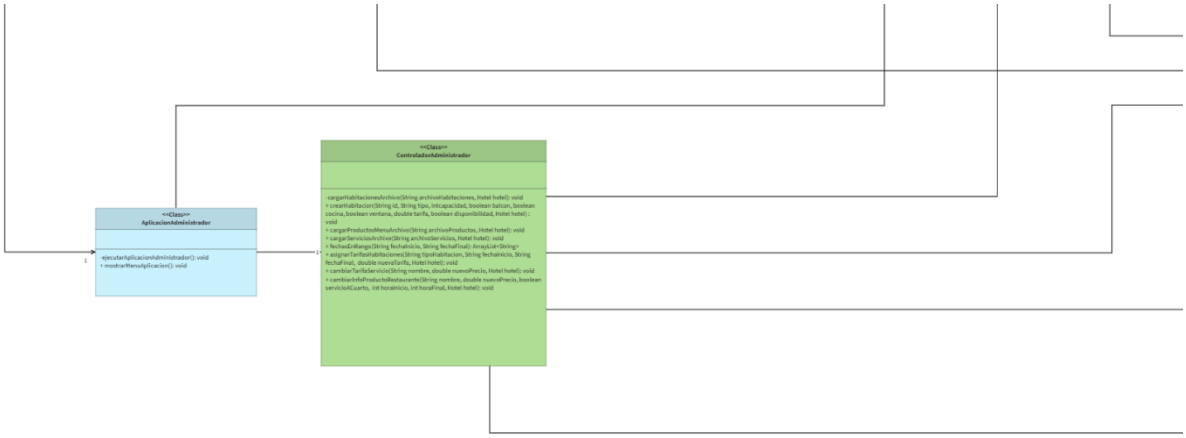


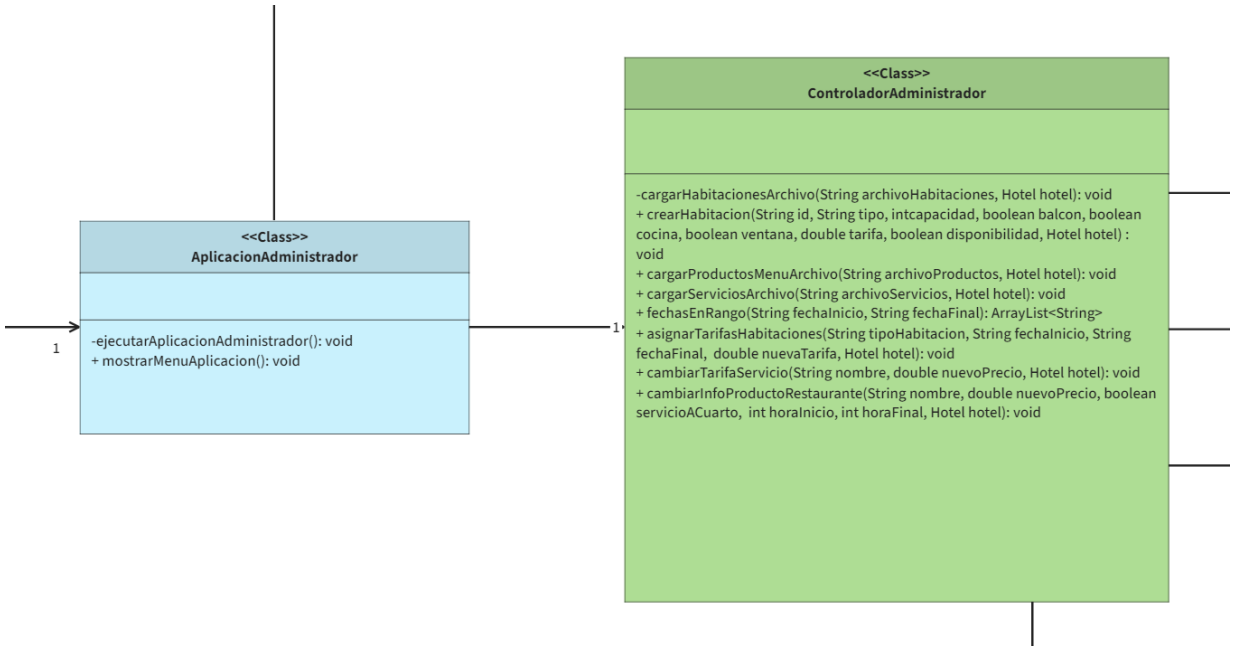


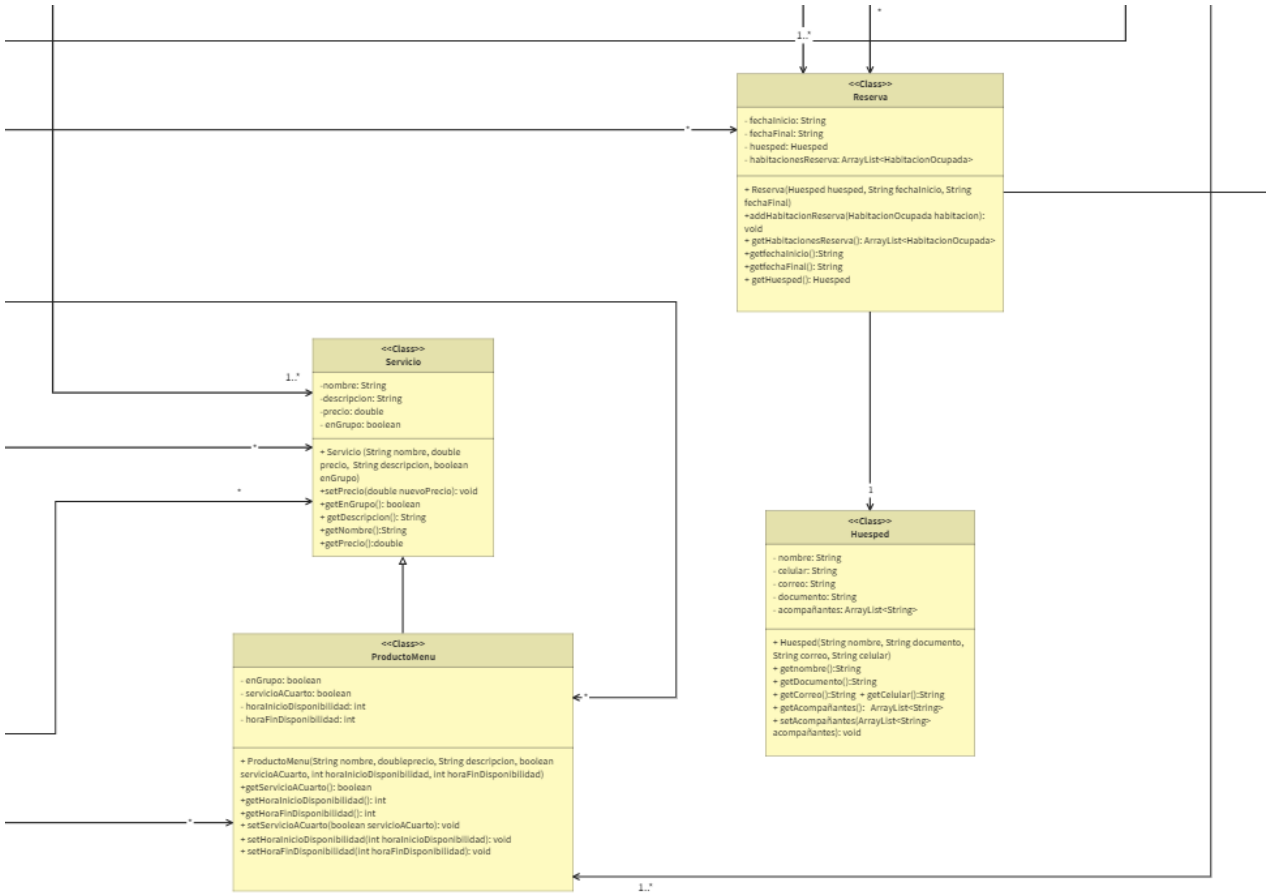

«Class»
Usuario
login: String
password: String
cargo: String
+ Usuario(String login, String password, String cargo)
+ getCargo(): String
+ verificarIdentificacion (String contrasenia): boolean
«Class»
ControladorRecepcionista
+fechasEnRango(String fechaInicio, String fechaFinal): ArrayList<String>
+cancelarReserva(Hotel hotel, String documento, String fechaActual, String fechaInicio, String fechaFinal): String
+ crearReserva(Hotel hotel, String nombre, String documento, String correo, String celular, String fechaInicio, String fechaFinal, ArrayList<String> acompañantes): void
+habitacionDisponible(Hotel hotel, String fechaInicio, String fechaFinal, String tipoHabitacion, String documentoHuesped): HabitacionOcupada
+ ocuparHabitacion(Hotel hotel, String fechaInicio, String fechaFinal, String tipoHabitacion, String documentoHuesped): HabitacionOcupada
+ agregarHabitacionAReserva(Reserva reserva, HabitacionOcupada habOcup): void
+ consumoHabitacionesReserva(Reserva reserva) : double
+ generarFacturaFinal(Reserva reserva, Hotel hotel): String
+ desocuparHabitaciones(Hotel hotel, HabitacionOcupada habOcup): void
+ consultarHabitacionesDisponibles(Hotel hotel): Map<String, Map<String, Habitacion>>
+ consultarHabitacionesOcupadas(Hotel hotel): Map<String, Map<String, ArrayList<HabitacionOcupada>>>
«Class»
AplicacionRecepcionista
+ejecutarAplicacionRecepcionista(): void
+ mostrarMenuAplicacion(): void
«Class»
Hotel
usuarios: Map<String, Usuario>
servicios: Map<String, Servicio>
habitacionesDisponibles: Map<String, Map<String, Habitacion>>
habitacionesOcupadas: Map<String, Map<String, ArrayList<HabitacionOcupada>>>
menu: Map<String, ProductoMenu>
reservas: Map<String, ArrayList<Reserva>>
modificacionesTarifasHabitaciones: Map<String, Double>
+ Hotel()
+getHabitacionesDisponiblesHotel(): Map<String, Map<String, Habitacion>>
+getHabitacionesOcupadasHotel(): Map<String, Map<String, HabitacionOcupada>>
+ getServiciosHotel():Map<String, Servicio>
+ getMenuHotel(): Map<String, ProductoMenu>
+ getReservas(): Map<String, ArrayList<Reserva>>
+ getUsuarios(): Map<String, Usuario>
+ getModificacionesHabitaciones(): Map<String, Map<String, Double>>
«Class»
ControladorEmpleado

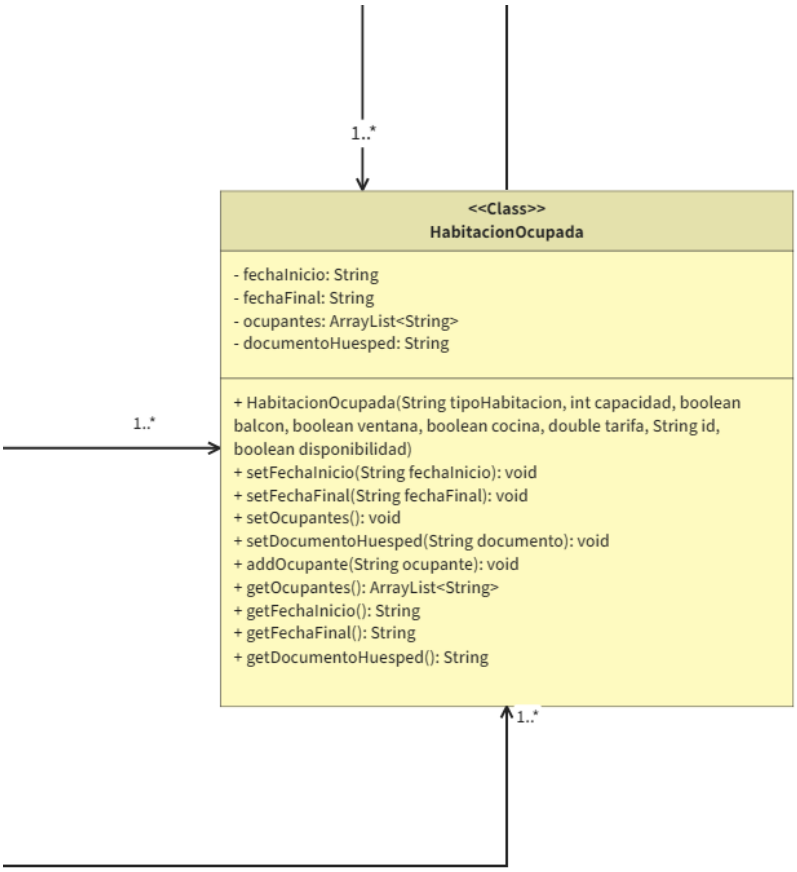

«Class»
AplicacionRecepcionista
+ejecutarAplicacionRecepcionista(): void
+ mostrarMenuAplicacion(): void
1
1
«Class»
ControladorRecepcionista
+fechasEnRango(String fechaInicio, String fechaFinal): ArrayList<String>
+cancelarReserva(Hotel hotel, String documento, String fechaActual, String fechaInicio, String fechaFinal): String
+ crearReserva(Hotel hotel, String nombre, String documento, String correo, String celular, String fechaInicio, String fechaFinal, ArrayList<String> acompañantes): void
+habitacionDisponible(Hotel hotel, String fechaInicio, String fechaFinal, String tipoHabitacion, String documentoHuesped): HabitacionOcupada
+ ocuparHabitacion(Hotel hotel, String fechaInicio, String fechaFinal, String tipoHabitacion, String documentoHuesped): HabitacionOcupada
+ agregarHabitacionAReserva(Reserva reserva, HabitacionOcupada habOcup): void
+ consumoHabitacionesReserva(Reserva reserva) : double
+ generarFacturaFinal(Reserva reserva, Hotel hotel): String
+ desocuparHabitaciones(Hotel hotel, HabitacionOcupada habOcup): void
+ consultarHabitacionesDisponibles(Hotel hotel): Map<String, Map<String, Habitacion>>
+ consultarHabitacionesOcupadas(Hotel hotel): Map<String, Map<String, ArrayList<HabitacionOcupada>>>
1


<<Class>>
Hotel
-usuarios: Map<String, Usuario>
- servicios: Map<String, Servicio> - habitacionesDisponibles: Map<String, Map<String, Habitacion> >
- habitacionesOcupadas: Map<String, Map<String, ArrayList<HabitacionOcupada>>>
-menu: Map<String, ProductoMenu>
- reservas: Map<String, ArrayList<Reserva>>
- modificacionesTarifasHabitaciones: Map<String, Double>
+ Hotel()
+getHabitacionesDisponiblesHotel(): Map<String, Map<String, Habitacion>>
+getHabitacionesOcupadasHotel(): Map<String, Map<String, HabitacionOcupada>>
+ getServiciosHotel():Map<String, Servicio>
+ getMenuHotel(): Map<String, ProductoMenu>
+ getReservas(): Map<String, ArrayList<Reserva>>
+ getUsuarios(): Map<String, Usuario>
+ getModificacionesHabitaciones(): Map<String, Map<String, Double>>
1
1
1
1
1
1
<<class>>
Usuario
- logIn: String
- password: String
- cargo: String
+ Usuario(String logIn, String password, String cargo)
+ getCargo(): String
+ verificarIdentificacion (String contrasenia): boolean
1..*


<<Class>>
AplicacionAdministrador
-ejecutarAplicacionAdministrador(): void
+ mostrarMenuAplicacion(): void
<<Class>>
ControladorAdministrador
-cargarHabitacionesArchivo(String archivoHabitaciones, Hotel hotel): void
+ crearHabitacion(String id, String tipo, intcapacidad, boolean balcon, boolean cocina, boolean ventana, double tarifa, boolean disponibilidad, Hotel hotel) : void
+ cargarProductosMenuArchivo(String archivoProductos, Hotel hotel): void
+ cargarServiciosArchivo(String archivoServicios, Hotel hotel): void
+ fechasEnRango(String fechaInicio, String fechaFinal): ArrayList<String>
+ asignarTarifasHabitaciones(String tipoHabitacion, String fechaInicio, String fechaFinal, double nuevaTarifa, Hotel hotel): void
+ cambiarTarifaServicio(String nombre, double nuevoPrecio, Hotel hotel): void
+ cambiarInfoProductoRestaurante(String nombre, double nuevoPrecio, boolean servicioACuarto, int horaInicio, int horaFinal, Hotel hotel): void


<<Class>>
AplicacionAdministrador
-ejecutarAplicacionAdministrador(): void
+ mostrarMenuAplicacion(): void
1
1
<<Class>>
ControladorAdministrador
-cargarHabitacionesArchivo(String archivoHabitaciones, Hotel hotel): void
+ crearHabitacion(String id, String tipo, intcapacidad, boolean balcon, boolean cocina, boolean ventana, double tarifa, boolean disponibilidad, Hotel hotel) : void
+ cargarProductosMenuArchivo(String archivoProductos, Hotel hotel): void
+ cargarServiciosArchivo(String archivoServicios, Hotel hotel): void
+ fechasEnRango(String fechaInicio, String fechaFinal): ArrayList<String>
+ asignarTarifasHabitaciones(String tipoHabitacion, String fechaInicio, String fechaFinal, double nuevaTarifa, Hotel hotel): void
+ cambiarTarifaServicio(String nombre, double nuevoPrecio, Hotel hotel): void
+ cambiarInfoProductoRestaurante(String nombre, double nuevoPrecio, boolean servicioACuarto, int horaInicio, int horaFinal, Hotel hotel): void


1..*
*
<<Class>>
Reserva
- fechaInicio: String
- fechaFinal: String
- huesped: Huesped
- habitacionesReserva: ArrayList<HabitacionOcupada>
+ Reserva(Huesped huesped, String fechaInicio, String fechaFinal)
+addHabitacionReserva(HabitacionOcupada habitacion): void
+ getHabitacionesReserva(): ArrayList<HabitacionOcupada>
+getfechaInicio():String
+getfechaFinal(): String
+ getHuesped(): Huesped
<<Class>>
Servicio
-nombre: String
-descripcion: String
-precio: double
- enGrupo: boolean
+ Servicio (String nombre, double precio, String descripcion, boolean enGrupo)
+setPrecio(double nuevoPrecio): void
+getEnGrupo(): boolean
+ getDescripcion(): String
+getNombre():String
+getPrecio():double
1..*
*
*
1
<<Class>>
Huesped
- nombre: String
- celular: String
- correo: String
- documento: String
- acompañantes: ArrayList<String>
+ Huesped(String nombre, String documento, String correo, String celular)
+ getnombre():String
+ getDocumento():String
+ getCorreo():String + getCelular():String
+ getAcompañantes(): ArrayList<String>
+ setAcompañantes(ArrayList<String> acompañantes): void
<<Class>>
ProductoMenu
- enGrupo: boolean
- servicioACuarto: boolean
- horaInicioDisponibilidad: int
- horaFinDisponibilidad: int
+ ProductoMenu(String nombre, doubleprecio, String descripcion, boolean servicioACuarto, int horaInicioDisponibilidad, int horaFinDisponibilidad)
+getServicioACuarto(): boolean
+getHoraInicioDisponibilidad(): int
+getHoraFinDisponibilidad(): int
+ setServicioACuarto(boolean servicioACuarto): void
+ setHoraInicioDisponibilidad(int horaInicioDisponibilidad): void
+ setHoraFinDisponibilidad(int horaFinDisponibilidad): void
*
*
1..*



Decisiones de diseño:

Se han implementado diversas decisiones al diseño de la solución. Entra ellas están el número de restaurantes en la aplicación, el tener distintos controladores y aplicaciones para cada tipo de usuario, o el uso de archivos .txt para el manejo de la información.

En primer lugar, se decidió crear un solo restaurante dentro de la aplicación para el hotel a pesar de que en la realidad cada hotel puede tener más de un hotel. Es decir, la aplicación mostrará todos los productos de los menús de todos los restaurantes. Lo anterior se da por facilidad de implementación con el valor agregado que desde cualquier restaurante del hotel se pueden pedir y facturar productos de cualquier otro restaurante.

Por otro lado, se decidió que cada tipo de usuario tendría su propia aplicación y controlador. Lo anterior permite que mayor facilidad en el sistema de autenticación de usuario y evita que un usuario no administrador tenga acceso a funcionalidades de administrador, por ejemplo. Lo anterior dificulta un poco la implementación, especialmente si se añaden más tipos de usuarios. Sin embargo, esta decisión facilita la autenticación de usuarios y la seguridad del sistema.

Finalmente se tiene la decisión del uso de archivos .txt que se da para facilitar la carga de los archivos por parte del administrador. Este tipo de archivos son de baja complejidad para el administrador y evitan la necesidad de desarrollar un sistema binario de archivos, que puede llegar a ser muy complejo.

PERSISTENCIA

Para garantizar que la información que se genere en cada ejecución de la aplicación quede guardada en la base de datos, de modo que los datos persistan a lo largo del tiempo, se utilizarán archivos de texto en el formato .txt para almacenar y cargar la información. El motivo por el que se usarán archivos .txt es que su manipulación es bastante sencilla y porque podremos darle a la información el formato que más se ajuste a nuestras necesidades.

La información que decidimos guardar en estos archivos fue la que corresponde a las reservas activas del hotel, las habitaciones disponibles del hotel, las habitaciones ocupadas en ciertas fechas, los productos del menú del restaurante (esto tiene en cuenta las modificaciones realizadas por los administradores del hotel), los servicios ofrecidos por el hotel (también tiene en cuenta las modificaciones efectuadas por los administradores) y las modificaciones que se le hagan a las tarifas de cada uno de los tipos de habitaciones para unas fechas dadas.

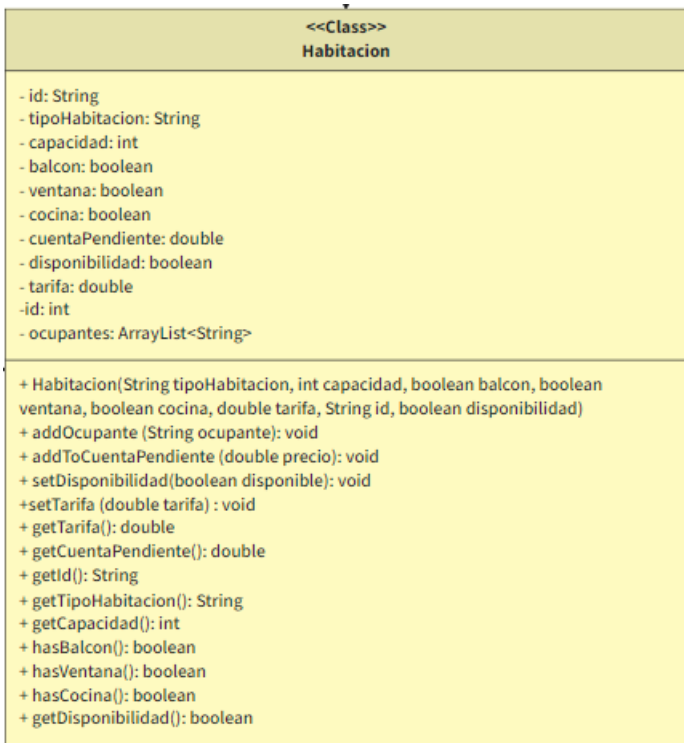
Cada una de estas estará almacenada en archivos independientes ubicados en una carpeta diferente, una carpeta externa, a dónde se encuentra la implementación de la aplicación para evitar modificaciones por parte de los usuarios de la aplicación (Administradores, recepcionistas y otros empleados).

La manera en la que hará la manipulación de estos archivos será a través de un controlador encargado exclusivamente de estas funcionalidades, el cual cargará información pasada y la actualizará con los cambios que se realicen en cada ejecución de la aplicación. De este modo disminuiremos el acoplamiento entre los controladores correspondientes a los usuarios de la aplicación y el controlador de persistencia.

DESCRIPCIÓN DETALLA (RESPONSABILIDAD Y JUSTIFICACIÓN) DE LOS COMPONENTES DEL DISEÑO QUE COMPONEN LAS DIFERENTES FUNCIONALIDADES DEL SISTEMA

MODEL

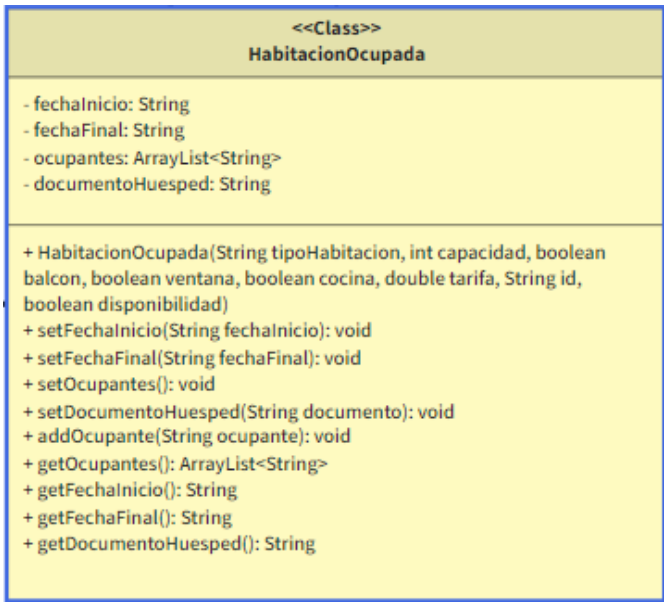
CLASS HABITACIÓN



RESPONSABILIDAD Y JUSTIFICACIÓN DE CREACIÓN DE LA CLASE

La clase Habitación se establece con el fin de crear y administrar (los diferentes atributos con los que cuenta) por separado cada una de las habitaciones (objetos de la clase) de las que estará compuesto el hotel. Esto se hará por medio de los atributos y métodos asignados (como los podemos ver en la imagen), que nos permitirán, primero que todo, acceder al valor de ciertos atributos por medio de los get y los has presentes. Así mismo, se cuenta con un constructor, que nos permitirá crear un objeto de tipo habitación; algunos set, que nos facilitan la cambiar el valor de algunos atributos de las habitaciones, como es su disponibilidad y tarifa, y; algunos add, que nos permiten agregar información a atributos como “ocupantes” (ocupantes de la habitación) y “cuentaPendiente” (se refiere a la cuenta que se le asigna a una habitación cuando se realizan algunos consumos que no son pagados de forma inmediata).

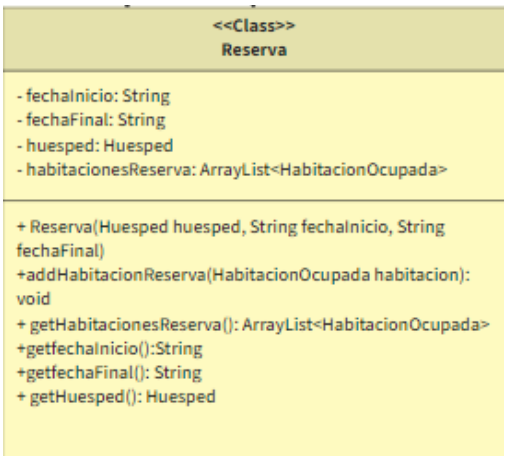
CLASS HABITACIÓN OCUPADA



RESPONSABILIDAD Y JUSTIFICACIÓN DE CREACIÓN DE LA CLASE

La clase Habitación Ocupada extiende de la clase Habitación y se establece con el fin de crear y administrar por separado cada una de las habitaciones ocupadas (objetos de la clase) de las que estará compuesto el hotel. Esto se hará por medio de los atributos y métodos asignados (como los podemos ver en la imagen), que nos permiten, con el constructor, crear un objeto de tipo habitación ocupada; con los set, cambiar el valor de algunos atributos de las habitaciones, como es su fecha de inicio y fin de ocupación, sus ocupantes y el documento del responsable de la habitación ocupada; con los add, agregar información a atributos como “ocupante” (ocupantes de la habitación), y; con los gets, acceder al valor actualizado de los atributos de la clase.

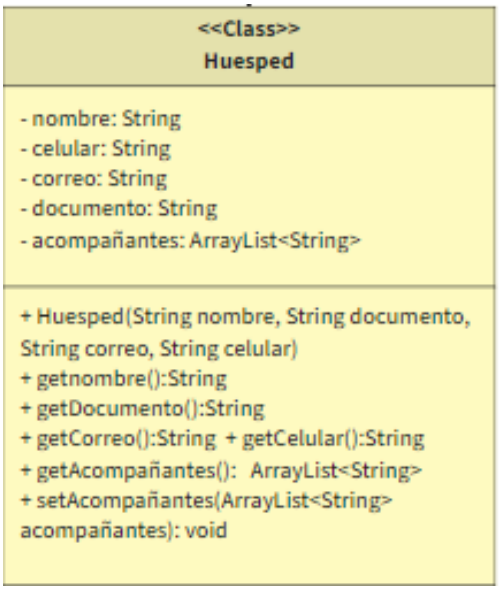
CLASS RESERVA



RESPONSABILIDAD Y JUSTIFICACIÓN DE CREACIÓN DE LA CLASE

La clase Reserva, se establece con el fin de crear y administrar por separado las reservas que tenga y pueda llegar a tener el hotel. De esta forma, esta clase nos facilitará conocer la fecha de inicio y fin de una reserva, las habitaciones de las que se compone la reserva y que serán ocupadas (las cuales pueden ser agregadas a disposición del huésped), y el responsable de la misma.

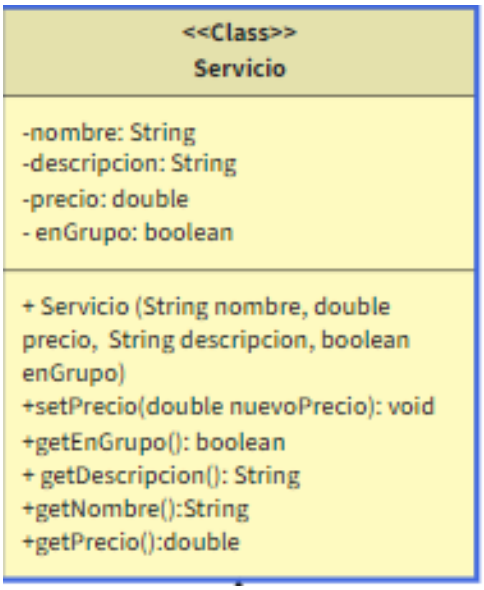
CLASS HUESPED



RESPONSABILIDAD Y JUSTIFICACIÓN DE CREACIÓN DE LA CLASE

Esta clase se establece con el fin de crear el objeto Huésped (que en otras palabras es un cliente del hotel), que será usado para el registro de los ocupantes del hotel y de cada una de las habitaciones. Así mismo, será utilizado en el registro y asignación de consumos, servicios y cuentas pendientes, que es fundamental para conocer el responsable de consumo y uso de los diferentes servicios que ofrece el hotel.

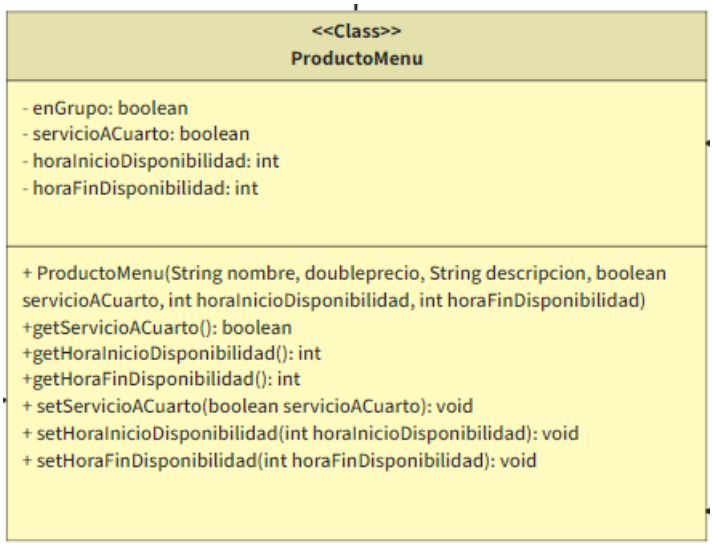
CLASS SERVICIO



RESPONSABILIDAD Y JUSTIFICACIÓN DE CREACIÓN DE LA CLASE

Esta clase se establece con el fin de crear y administrar cada uno de los servicios y consumos con los que cuenta el Hotel. Es así como, por medio de esta clase, se nos facilitará el acceso y la comunicación de la información, pues podremos conocer las características de cada servicio/ consumo como es su nombre, descripción, precio y si es de pago en grupo o por persona.

CLASS PRODUCTO MENÚ

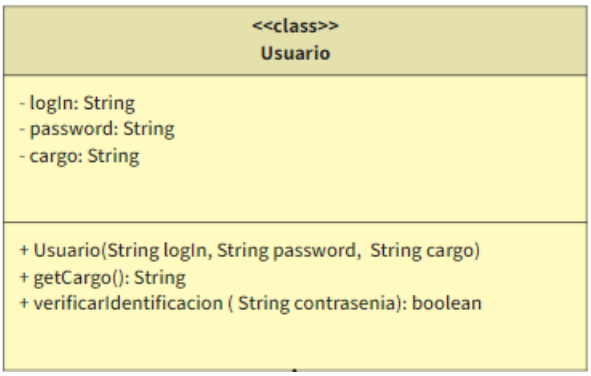


RESPONSABILIDAD Y JUSTIFICACIÓN DE CREACIÓN DE LA CLASE

Esta clase extiende de la clase Servicio y se establece con el fin de crear y administrar los diferentes productos de los que estará compuesto del menú que se ofrecerá en el restaurante

del hotel. De esta forma, esta clase nos facilitará el acceso y la comunicación de información, ya que nos permitirá conocer y modificar las características de cada uno de estos productos, como su hora inicio y fin de disponibilidad, y si están ponibles para ser consumidos en el cuarto.

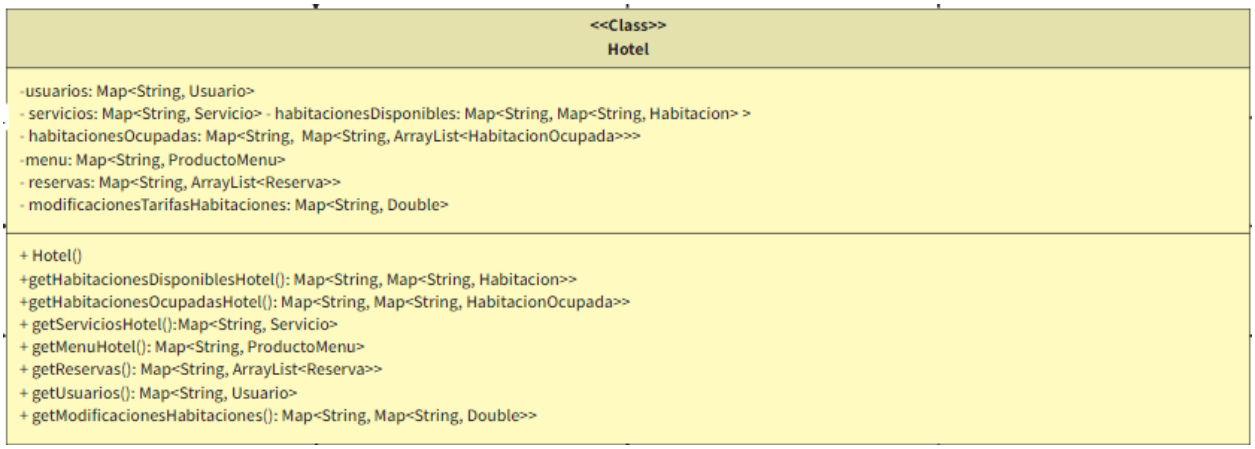
CLASS USUARIO



RESPONSABILIDAD Y JUSTIFICACIÓN DE CREACIÓN DE LA CLASE

Esta clase se establece con el fin de crear los usuarios del sistema construido. De esta forma, esta clase nos permitirá construir un usuario, que tendrá un login, una contraseña y un cargo, que será utilizado posteriormente para redirigirlo al menú de opciones correspondiente a su cargo. Igualmente, la clase Usuario, nos permitirá hacer un control de acceso y seguridad al sistema, ya que nos facilitará hacer la verificación de contraseña cuando un usuario quiera ingresar y ejecutar acciones permitidas según su cargo.

CLASS HOTEL



ACLARACIÓN DE LOS ATRIBUTOS

Usuarios: Map<Login del usuario, objeto de la clase usuario>

Servicios: Map<Nombre del servicio, objeto de la clase servicio>

HabitacionesOcupadas: Map<Tipo de Habitación, Map<Id de habitación, ArrayList<registro de las habitaciones ocupadas (una habitación puede estar ocupada en diferentes fechas)>>>

HabitacionesDisponibles: Map<Tipo de Habitación, Map<Id de habitación, ArrayList<registro de las habitaciones ocupadas (una habitación puede estar ocupada en diferentes fechas)>>>

Menu: Map<Nombre del producto, objeto de la clase ProductoMenu>

Reservas: Map<documento del huésped que hace la reserva, <objeto de la clase Reserva>

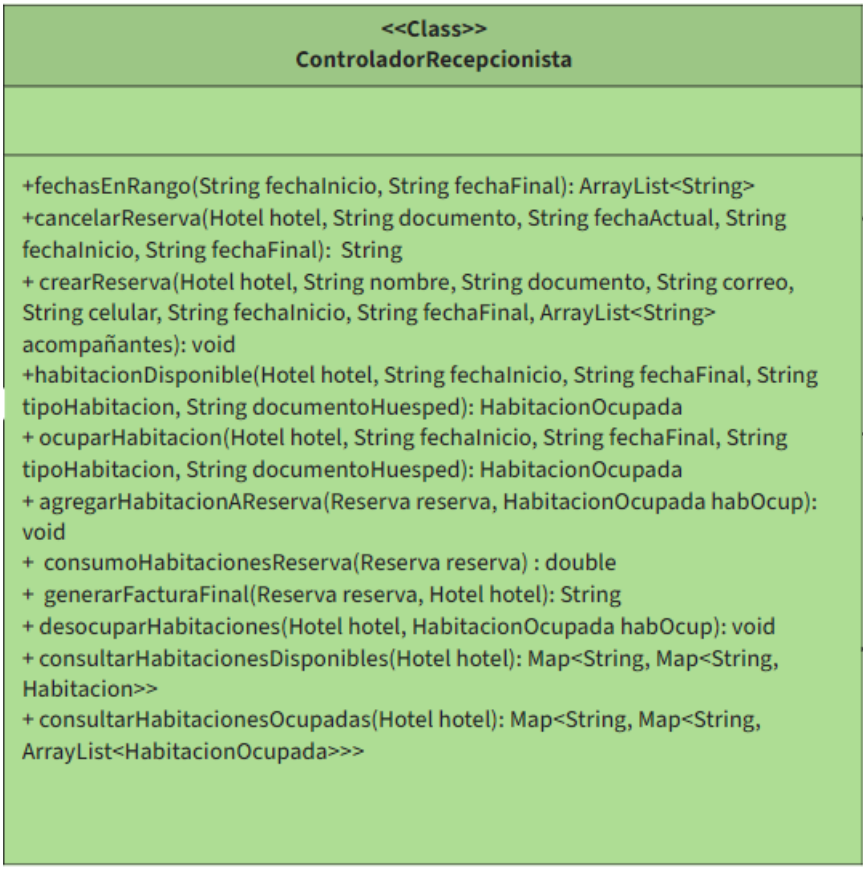
modificaciones Tarifas Habitaciones: Map<Tipo de habitación, Map<fecha, tarifa asignada por el administrador>>

RESPONSABILIDAD Y JUSTIFICACIÓN DE CREACIÓN DE LA CLASE

La clase Hotel, puede ser considerada como una de las más importantes en esta etapa, ya que es con la que conectan todas las clases y el eje central del sistema. Es así como, esta clase se establece con el fin de crear el Hotel, almacenar y acceder a toda la información del hotel, que se compone de las habitaciones, habitaciones ocupadas, reservas, menú, usuarios y las modificaciones de las tarifas de las habitaciones. Así, podemos ver como esta clase actúa como el gran INFORMATION HOLDER de la aplicación, siendo eje primordial para su funcionamiento.

CONTROLADOR

CLASS CONTROLADOR RECEPCIONISTA



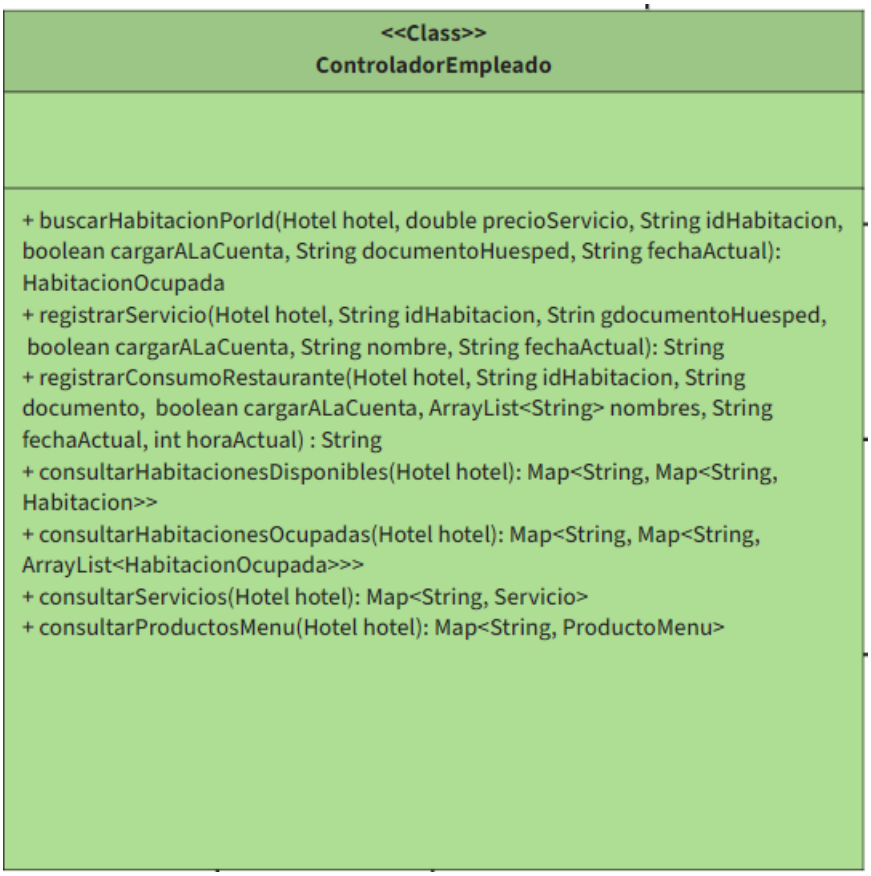
RESPONSABILIDAD Y JUSTIFICACIÓN DE CREACIÓN DE LA CLASE

Esta clase se establece con el objetivo de implementar la ejecución de cada una de las opciones con las que cuenta el menú disponible para el recepcionista del hotel. De esta forma, tenemos a los métodos más importantes que nos permiten comprender como se ejecuta cada una de las funcionalidades disponibles para un recepcionista:

- El método fechasEnRango, se encarga a partir de un rango de fechas en formato “DD/MM” determinar el conjunto de fechas que está dentro del mismo. Esto nos servirá, para estalecer las fechas de reserva, disponibilidad y ocupación de las diferentes habitaciones con las que cuenta el hotel.
- El método cancelarReserva, se encarga de cancelar una reserva a partir del documento del huésped responsable de una reserva, una fecha actual (para conocer los días de anticipación con los que se está realizando la cancelación) y de la fecha de inicio y fin de la reserva que se quiere cancelar. Este método, nos permite garantizar, por medio del uso de la fecha actual y la fecha de inicio, que para realizar una cancelación, un huésped debe realizarlo antes de las 48 horas antes de que inicie la reserva.

- El método habitacionDisponible y ocuparHabitación, se encarga de verificar si en un rango de fechas y para un tipo de habitación, existe una habitación disponible para reservar. Si es el caso, entonces ocupará la primera habitación disponible. Sin embargo, aún si no hay habitación disponible, verificará dentro de las habitaciones ocupadas si hay alguna que no se cruce con la fecha dispuesta de reserva, y si no lo hace la reservará y la ocupará, actualizando los datos del huésped (documentoHuesped) responsable al que se le asignará la habitación.
- El método agregarHabitaciónAReserva, se encarga de agregar una habitación a una reserva recibida por parámetro.
- El método generarFacturaFinal, se encarga de contabilizar el consumo por cada habitación perteneciente a una reserva recibida por parámetro (sumando la tarifa por noche y la cuenta pendiente asignada) y así obtener un saldo total, que será reflejado en la factura junto con el nombre del huésped responsable, la fecha de inicio y fin de la reserva, el número de noches y, el número de identificación de cada habitación ocupada.
- El método desocuparHabitaciones, se encarga de desocupar una habitación recibida por parámetro y ponerla nuevamente disponible para su reserva y ocupación.

CLASS CONTROLADOR EMPLEADO

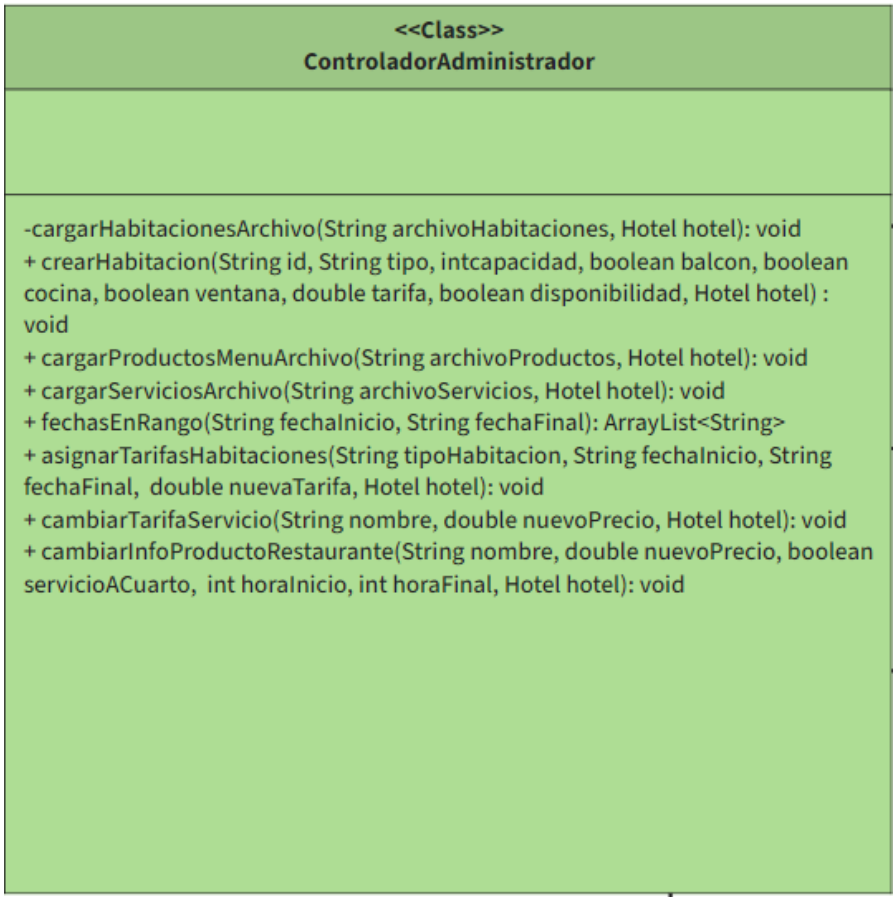


RESPONSABILIDAD Y JUSTIFICACIÓN DE CREACIÓN DE LA CLASE

Esta clase se establece con el objetivo de implementar la ejecución de cada una de las opciones con las que cuenta el menú disponible para un empleado del hotel. De esta forma, tenemos a los métodos más importantes que nos permiten comprender como se ejecuta cada una de las funcionalidades disponibles para un recepcionista:

- El método registrarServicio junto con el método buscarHabitacionPorId, se encarga de registrar un servicio disponible en el hotel, a una habitación que, por medio del método buscarHabitacionPorId, se encarga de buscarla, y si es el caso que el huésped no quiera pagar de forma inmediata el servicio consumido, se lo cargará a una cuenta pendiente. Por otro lado, así el pago se haga de forma inmediata o no, se le generará una factura al huésped/des en donde se registra la información pertinente del servicio consumido.
- El método registrarConsumoRestaurante junto con el método buscarHabitacionPorId, se encarga de registrar el consumo de algún producto de Restaurante, a una habitación que, por medio del método buscarHabitacionPorId, nos permite buscarla, y si es el caso que el huésped no quiera pagar de forma inmediata el servicio consumido, se lo cargará a una cuenta pendiente. Así mismo, para el registro del consumo, contabilizará, si el producto está dentro de la hora de disponibilidad, el precio de cada producto consumido (ProductoMenu) y por último, así el pago se haga de forma inmediata o no, se le generará una factura al huésped/des en donde se registra la información pertinente de cada producto consumido y del precio total del servicio.
- El método consultarHabitacionesDisponibles se encarga de retornar el mapa con la información de las habitaciones disponibles en el hotel.
- El método consultarHabitacionesOcupadas se encarga de retornar el mapa con la información de las habitaciones ocupadas en el hotel.
- El método consultarServicios se encarga de retornar el mapa con la información de servicios disponibles en el hotel.
- El método consultarProductosMenu se encarga de retornar el mapa con la información de los productosMenú disponibles en el restaurante del hotel.

CLASS CONTROLADOR ADMINISTRADOR



RESPONSABILIDAD Y JUSTIFICACIÓN DE CREACIÓN DE LA CLASE

Esta clase se establece con el objetivo de implementar la ejecución de cada una de las opciones con las que cuenta el menú disponible para el administrador del hotel. De esta forma, tenemos a los métodos más importantes que nos permiten comprender como se ejecuta cada una de las funcionalidades disponibles para un recepcionista:

- El método cargarHabitacionesArchivo se encarga de leer un archivo .txt en el que se almacena la información de las habitaciones. Es así como, cada línea del archivo constituye una habitación que pasará a ser un objeto de la clase habitación y a hacer parte del mapa de habitaciones disponibles. El archivo .txt está compuesto de esta forma:

  Id; tipo de habitación (s: suite, e: estándar, sd: suite doble); capacidad; balcón (boolean); ventana (boolean); cocina (boolean); tarifa; disponibilidad (boolean)

- El método crearHabitacion se encarga de crear una habitación una por una de forma directa sin necesidad de leer un archivo.

- El método cargarProductosMenuArchivo se encarga de leer un archivo .txt en el que se almacena la información de las productos del menú del restaurante del hotel. Es así como, cada línea del archivo constituye un producto del menú que pasará a ser un objeto de la clase ProductoMenu y a hacer parte del mapa de menú. El archivo .txt está compuesto de esta forma:

  nombre; costo; capacidad; descripción; servicio al Cuarto (boolean); horaInicio; horaFin

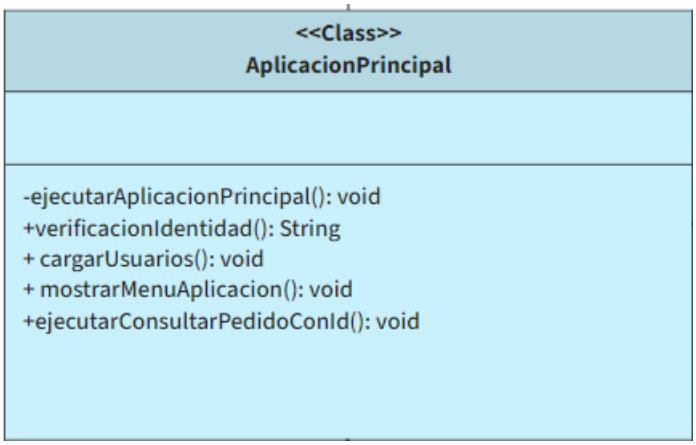
- El método cargarServiciosArchivo se encarga de leer un archivo .txt en el que se almacena la información de los servicios disponibles en el hotel. Es así como, cada línea del archivo constituye un servicio que pasará a ser un objeto de la clase Servicio y a hacer parte del mapa de servicios. El archivo .txt está compuesto de esta forma:

  nombre; costo; grupo (boolean); descripción

- El método asignarTarifasHabitaciones se encarga actualizar la tarifa (ingresada por parámetro) de las habitaciones correspondientes a un tipo de habitación (ingresado por parámetro) durante un rango de fechas (ingresadas por parámetro)
- El método cambiarTarifaServicio se encarga de actualizar el precio (ingresado por parámetro) de un servicio ingresado por parámetro, que es buscado por medio de algunos get.
- El método cambiarInfoProductoRestaurante se encarga de actualizar la información de un productoMenu ingresado por parámetro.

CONSOLA

CLASS APLICACIÓN PRINCIPAL



RESPONSABILIDAD Y JUSTIFICACIÓN DE CREACIÓN DE LA CLASE

Esta clase se establece con el objetivo de crear la interfaz principal con la que interactúan todos los usuarios del sistema y por medio de la cual son redirigidos a aplicaciones, que se adaptan a las funcionalidades disponibles para cada usuario dependiendo su cargo. De esta forma, los métodos más importantes de esta clase son:

- verificacionIdentidad: se encarga de verificar la identidad de cualquier persona que quiera ingresar al sistema. Esto lo hace accediendo al mapa en donde están almacenados todos los usuarios registrados en el sistema y utilizando el método verificarIdentificacion de la clase Usuario, que lo que hace es verificar la contraseña ingresada con la que está registrada. De esta forma, este método al final, si el usuario fue verificado devuelve un mensaje con el cargo de este o si no, muestra un error.
- cargarUsuarios: lee un archivo .txt en el que cada línea está compuesta de la información de un usuario. Cada línea está compuesta así:

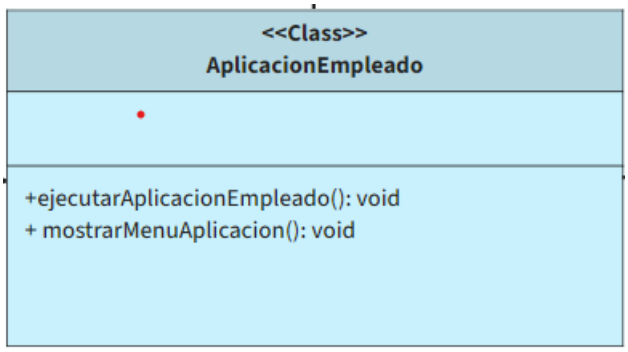
  logIn; password; cargo (R: recepcionista; A: administrador; E: empleado)

- mostrarMenuAplicacion: muestra las opciones iniciales del sistema:

  1. Ingresar al sistema
  2. Cerrar aplicación

- ejecutarAplicacion: se encarga de ejecutar toda la aplicación principal, llamando los métodos anteriores, para mostrar el menú principal y utilizar la verificación de Identidad para conocer el cargo de los usuarios que sean aprobados, para luego ser redirigidos a sus respectivas aplicaciones.

CLASS APLICACIÓN EMPLEADO



RESPONSABILIDAD Y JUSTIFICACIÓN DE CREACIÓN DE LA CLASE

Esta clase se establece con el objetivo de crear la interfaz principal con la que interactúan todos los empleados del hotel y por medio de la cual se muestran las opciones que se adaptan a las funcionalidades disponibles para su cargo. De esta forma, los métodos más importantes de la clase controlador Empleado que se relacionan con cada opción son:

Menú de opciones

```
"1. Registrar servicio"
"2. Registrar consumo en restaurante"
"3. Consultar habitaciones disponibles"
"4. Consultar habitaciones ocupadas"
"5. Consultar servicios del hotel"
"6. Consultar productos del menú"
"7. Cerrar aplicación empleado"
```

Opción 1:

- controlador.registrarServicio()

Opción 2: Adicionalmente, en esta opción se le muestra una serie de opciones sobre los productos que se desean consumir del menú ofrecido en el restaurante.

- controlador.registrarConsumoRestaurante()

Opción 3:

- controlador.consultarHabitacionesDisponibles()

Opción 4:

- controlador.consultarHabitacionesOcupadas()

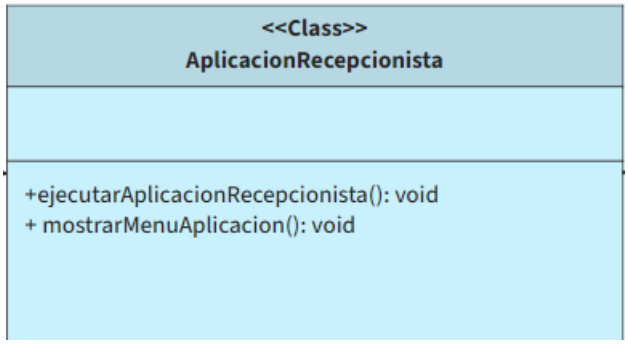
Opción 5:

- controlador.consultarServicios ()

Opción 6:

- controlador.consultarProductosMenu()

CLASS APLICACIÓN RECEPCIONISTA



RESPONSABILIDAD Y JUSTIFICACIÓN DE CREACIÓN DE LA CLASE

Esta clase se establece con el objetivo de crear la interfaz principal con la que interactúan todos los recepcionistas del hotel y por medio de la cual se muestran las opciones que se adaptan a las funcionalidades disponibles para su cargo. De esta forma, los métodos más importantes de la clase controlador Recepcionista que se relacionan con cada opción son:

Menú de opciones

```
"1. Crear reserva"
"2. Hacer check-out”
"3. Cancelar reserva"
"4. Consultar habitaciones disponibles"
"5. Consultar habitaciones ocupadas"
"6. Cerrar aplicación recepcionista"
```

Opción 1:

- controlador.crearReserva()

Opción 2:

- controlador.generarFacturaFinal()
- desocuparHabitaciones()

Opción 3:

- controlador.cancelarReserva()

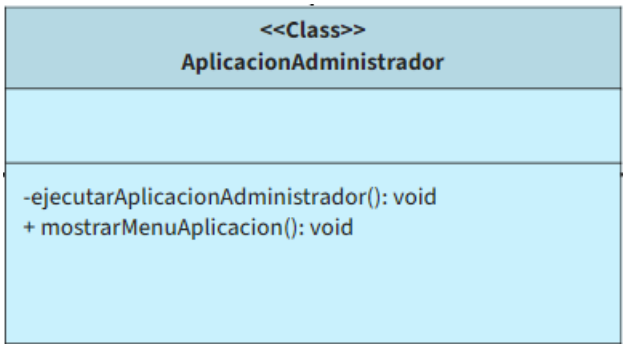
Opción 4:

- controlador.consultarHabitacionesDisponibles()

Opción 5:

- controlador. consultarHabitacionesOcupadas()

CLASS APLICACIÓN ADMINISTRADOR



RESPONSABILIDAD Y JUSTIFICACIÓN DE CREACIÓN DE LA CLASE

Esta clase se establece con el objetivo de crear la interfaz principal con la que interactúa el administrador y por medio de la cual se muestran las opciones que se adaptan a las funcionalidades disponibles para su cargo. De esta forma, los métodos más importantes de la clase controlador Recepcionista que se relacionan con cada opción son:

Menú de opciones

```
"1. Cargar habitaciones"
"2. Crear habitación"
"3. Cargar productos del menú"
"4. Cargar servicios del hotel"
"5. Asignar tarifa por tipo de habitación en rango de fechas"
"6. Cambiar tarifa de un servicio"
"7. Cambiar información de un plato/bebida del menú"
"8. Cerrar aplicación administrador"
```

Opción 1:

- controlador.cargarHabitacionesArchivo()

Opción 2:

- controlador.crearHabitacion()

Opción 3:

- controlador.cargarProductosMenuArchivo()

Opción 4:

- controlador.cargarServiciosArchivo()

Opción 5:

- controlador.asignarTarifasHabitaciones()

Opción 6:

- controlador. cambiarTarifaServicio()

Opción 7:

- controlador.cambiarInfoProductoRestaurante()